

# Elementi di Architettura e Sistemi Operativi

Bioinformatica - Tiziano Villa

6 Luglio 2012

Nome e Cognome:

Matricola:

Posta elettronica:

problema	punti massimi	i tuoi punti
problema 1	6	
problema 2	7	
problema 3	7	
problema 4	10	
totale	30	

1. Rispondere in modo preciso ma conciso alle seguenti domande.

Interpretando la paginazione su richiesta come una forma di cache tra memoria principale e disco, si risponda alle seguenti domande con una breve motivazione delle risposte.

Traccia di soluzione.

Si noti come premessa che l'esercizio non postula una cache tra la memoria principale e il disco, ma chiede d'interpretare la memoria principale come una cache del disco (cache virtuale se si vuole) le cui caratteristiche progettuali sono determinate dalla politica d'impaginazione.

(a) Qual e' la dimensione di un blocco ?

Traccia di soluzione.

Una pagina.

(b) Qual e' l'organizzazione della cache (diretta, associativa a insiemi, completamente associativa) ?

Traccia di soluzione.

Completamente associativa: associazione arbitraria da virtuale a fisico.

(c) Come si trova una pagina nella cache (cioe' in questo caso nella memoria principale) ?

Traccia di soluzione.

Si cerca nella TLB, e se non si trova si consulta la tavola delle pagine.

(d) Che cosa si fa in caso di mancata presenza di una pagina ?

Traccia di soluzione.

Si preleva dal disco la pagina mancante.

(e) Quali sono gli algoritmi di rimpiazzo di una pagina in caso di mancata presenza di essa ?

Traccia di soluzione.

Varianti di LRU.

(f) Qual e' la politica di aggiornamento del disco in caso di scrittura ?

Traccia di soluzione.

Si posticipa l'aggiornamento della pagina su disco al momento in cui la pagina e' eliminata dalla memoria centrale ("write-back"). Serve la cifra binaria di scrittura ("dirty bit").

2. Si consideri il seguente codice incompleto per scrivere le primitive di un semaforo  $P$  (*Wait*) and  $V$  (*Signal*) per un sistema uniprocessore.

```
typedef struct {
    int count;
} semaphore;

P(semaphore *S) {
    while (1) {
        disabilita le interruzioni;
        if (S->count ? ) {
            S->count ?;
            abilita le interruzioni
            return;
        }
        else abilita le interruzioni;
    }
}

V(semaphore *S) {
    disabilita le interruzioni;
    S->count ?;
    abilita le interruzioni
}
```

- (a) Si descriva brevemente che cos'è un semaforo e si mostri lo pseudo-codice della definizione classica delle operazioni  $P$  e  $V$ .

Traccia di soluzione.

Un semaforo è una variabile intera cui si può accedere, escludendo l'inizializzazione, solo tramite due operazioni atomiche predefinite:  $P$  (*Wait*) and  $V$  (*Signal*).

Le definizioni classiche di *Wait* e *Signal* in pseudo-codice sono le seguenti:

```
Wait (S) {  
    while (S <= 0)  
        ;  
    S--;  
}
```

```
Signal (S) {  
    S++;  
}
```

- (b) Si completi il codice iniziale sostituendo i "?" con le parti mancanti, e lo si commenti spiegando perché realizza correttamente le primitive di un semaforo.

Traccia di soluzione.

La disabilitazione e riabilitazione delle interruzioni sono sufficienti per ottenere l'atomicità delle operazioni in caso di processore singolo.

```
typedef struct {
    int count;
} semaphore;

P(semaphore *S) {
    while (1) {
        disabilita le interruzioni;
        if (S->count > 0 ) {
            S->count -= 1;
            abilita le interruzioni
            return;
        }
        else abilita le interruzioni;
    }
}

V(semaphore *S) {
    disabilita le interruzioni;
    S->count += 1;
    abilita le interruzioni
}
```

(c) Che problemi ha il codice precedente ?

Traccia di soluzione.

Attesa attiva in  $P$  per il ciclo  $while(1)$ , poiché quando il semaforo non è positivo il processo che esegue tale codice lo ripete inutilmente (per il tempo del processore assegnato al processo), anche se il semaforo può diventare positivo solo rilasciando il processore a un processo che esegua una  $V$ .

Non è equo in  $P$ .

(Stallo in  $P$  per certe priorità').

- (d) Si modifichi il codice come segue: si aggiunga al semaforo una coda di processi in attesa; se  $P$  fallisce si aggiunga il processo alla coda, con  $V$  si tolga il processo dalla coda se essa non e' vuota.

Si commenti la soluzione risultante rispetto a quella iniziale e ai problemi del punto precedente.

Traccia di soluzione.

```
typedef struct {
    int count;
    queue q;
} semaphore;

P(semaphore *S) {
    disabilita le interruzioni;
    if (S->count > 0) {
        S->count -= 1;
        abilita le interruzioni
        return;
    }
    else {
        aggiungi il processo alla coda S->q;
        abilita le interruzioni;
    }
}

V(semaphore *S) {
    disabilita le interruzioni;
    S->count += 1;
    if (la coda S->q non e' vuota) {
        risveglia il primo processo di S->q
    }
    abilita le interruzioni
}
```

Questa soluzione migliora quella iniziale perche' ad es. non presenta attesa attiva in  $P$ .

3. Un sistema operativo offre la memoria virtuale paginata, utilizzando un processore con una durata di ciclo di 1 microsecondo. L'accesso a una pagina in memoria diversa da quella corrente richiede un altro microsecondo. Le pagine hanno 1000 parole e lo strumento di paginazione e' un cilindro che ruota a 3000 giri al minuto e trasferisce un milione di parole al secondo.

Dal sistema si ottengono le seguenti misurazioni statistiche:

- L'1% di tutte le istruzioni eseguite hanno avuto accesso a una pagina diversa da quella corrente.
- L'80% delle istruzioni che hanno avuto accesso a una pagina diversa da quella corrente hanno avuto accesso a una pagina gia' in memoria.
- Quando e' stata richiesta una nuova pagina, la pagina da rimpiazzarsi era stata modificata nel 50% dei casi.

Calcolare (in microsecondi =  $\mu sec.$ ) il tempo effettivo di esecuzione delle istruzioni di questo sistema, assumendo che esso stia eseguendo un solo processo e che il processore sia inattivo durante i trasferimenti del cilindro.

Traccia.

Prima si calcoli il tempo di accesso al cilindro. Si noti che il cilindro si puo considerare come un tipo particolare di disco rigido per cui non si debba considerare il tempo di posizionamento della testina, cioe' per il cilindro si deve tener conto solo del tempo di rotazione e del tempo di trasferimento.

Dopo si calcoli il tempo per le istruzioni che accedono alla pagina corrente, quello per le pagine non correnti ma in memoria, quello per le pagine che devono essere prelevate dal cilindro, e infine quello per le pagine che devono essere riscritte sul cilindro (in quanto modificate in scrittura e quindi che devono essere di nuovo salvate sul cilindro). Da ultimo si sommino tutti i contributi.

Traccia di soluzione.

Si noti che un accesso al tamburo richiede un tempo di rotazione e un tempo di trasferimento:

- Tempo di rotazione (si sceglie di calcolare il tempo di una semirotaazione):  
3000 rotazioni al minuto =  $(3000/60)=50$  rotazioni al secondo =  
100 semirotaazioni al secondo  
una semirotaazione richiede  $1/100$  sec., cioè'  
 $10^4 10^{-4} (1/100) \text{ sec.} = 10^4 10^{-4} 10^{-2} \text{ sec.} = 10^4 10^{-6} \text{ sec.} = 10^4 \mu\text{sec.}$
- Tempo di trasferimento:  
si trasferiscono  $10^6$  parole al secondo =  $10^3$  pagine al secondo (1 pagina =  $10^3$  parole)  
il trasferimento di una pagina richiede  $1/1000$  sec., cioè'  
 $10^3 10^{-3} (1/1000) \text{ sec.} = 10^3 10^{-3} 10^{-3} \text{ sec.} = 10^3 10^{-6} \text{ sec.} = 10^3 \mu\text{sec.}$
- Tempo di rotazione + tempo di trasferimento:  
 $10^4 \mu\text{sec.} + 10^3 \mu\text{sec.}$

Tempo medio effettivo di esecuzione di un'istruzione:

- Il 99% delle istruzioni accedono alla pagina corrente e perciò' richiedono solo un microsecondo (ciclo del processore):

$$0,99 \times 1 \mu\text{sec.} = 0,99 \mu\text{sec.}$$

- L'80% delle istruzioni rimanenti accedono a un'altra pagina in memoria al costo addizionale di un altro microsecondo:

$$(80\% \times 0,01 =) 0,008 \times (1 + 1) \mu\text{sec.} = 0,008 \times 2 \mu\text{sec.} = 0,016 \mu\text{sec.}$$

- Il restante 20% delle istruzioni rimanenti richiedono un accesso al tamburo per prelevare una pagina:

$$(20\% \times 0,01 =) 0,002 \times (10^4 \mu\text{sec.} + 10^3 \mu\text{sec.}) = 22 \mu\text{sec.}$$

- La meta' del precedente restante 20% delle istruzioni rimanenti richiede una scrittura sul tamburo della pagina rimpiazzata (in quanto modificata):

$$((20\% \times 0,01)/2 =) 0,001 \times (10^4 \mu\text{sec.} + 10^3 \mu\text{sec.}) = 11 \mu\text{sec.}$$

- Sommando tutti i contributi si ha:

$$0,99 \mu\text{sec.} + 0,016 \mu\text{sec.} + 22 \mu\text{sec.} + 11 \mu\text{sec.} \approx 34 \mu\text{sec.}$$

4. Si progetti un circuito sequenziale che realizza la seguente specifica:

- Ci sono due segnali binari d'ingresso  $X_1X_2$  e un segnale binario d'uscita  $Z$ .
- Se  $X_2 = 0$ , l'uscita  $Z$  nel generico istante  $t$  vale  $Z(t) = X_1(t - 1)$ , se  $X_2 = 1$ , l'uscita  $Z$  nel generico istante  $t$  vale  $Z(t) = X_1(t - 2)$ .  
Sia 00 00 la sequenza di azzeramento (cioe', alla partenza del circuito si danno gl'ingressi 00 00).

(a) Si disegni il grafo delle transizioni di una macchina a stati finiti di tipo Mealy che corrisponde alla specifica. S'indichi lo stato iniziale.

(b) Si minimizzi il numero degli stati della macchina proposta, applicando l'algoritmo di minimizzazione degli stati.

(c) Si scriva la tavola delle transizioni con gli stati futuri e le uscite e la si codifichi.

- (d) Supponendo di usare bistabili di tipo D, si derivino le equazioni minimizzate di eccitazione degl'ingressi dei bistabili e le equazioni minimizzate delle uscite.

- (e) Si realizzi il circuito sequenziale corrispondente con bistabili di tipo D campionati sul fronte di salita, invertitori e porte NAND. Si etichettino con chiarezza i segnali.